Retour d'expérience sur l'utilisation d'Easybuild

C. Pera, L. Pouilloux, D. Calugaru, G. Berret, A. Cadiou, C. Petit, L. Taulelle

JCAD, Lyon, mercredi 24 octobre 2018

Fédération Lyonnaise de Modélisation et Sciences Numériques

Problématique

Contexte

- Gestion des bibliothèques applicatives en calcul scientifique et de leurs dépendances sur des systèmes optimisés

Problèmes

- pas de solution proposée, ou inadaptée (performance), par le système de gestion de paquet natif (RPM/DEB),
- complexités d'installations à partir des "sources" sur les systèmes HPC,
- difficultés pour gérer plusieurs versions d'un même logiciel en environnement multi-utilisateurs,
- disponibilité et reproductibilité sur différentes plateformes matérielles et logicielles
- complexité et rareté de l'expertise technique.

⇒ **Recherche d'un environnement communautaire(?) qui apporte, au moins partiellement, une solution.**

Solutions existantes

Quelques solutions proposant des logiciels dans leurs versions binaires :

- les systèmes de gestion de paquets des distributions,
- Conda,
- Nix, Guix.

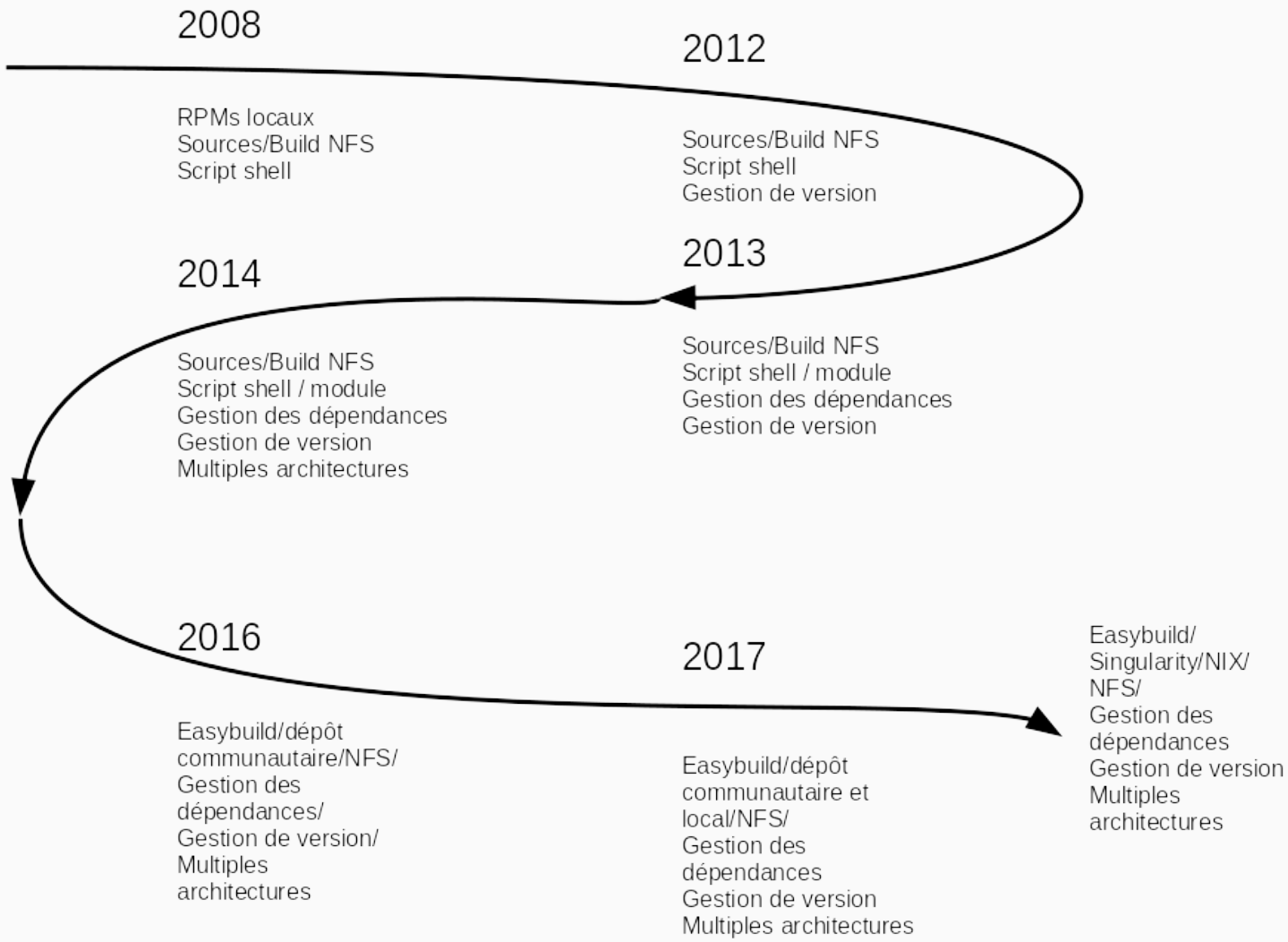
Des solutions proposant des containers ou machines virtuelles :

- Singularity, Shifter, CharlieCloud.

Les solutions offrant les sources et l'environnement pour la génération de versions binaires :

- Easybuild,
- Spack,
- Do It Yourself !

Le cheminement



Solution choisie

Easybuild

`https://easybuild.readthedocs.io/en/latest/`

- Une communauté importante **provenant de centres HPC** avec des contraintes proches des nôtres.
- Disponibilité de **logiciels de recherche** dans leurs versions les plus récentes.
- Génération de **binaires optimisés** pour les plateformes matériels ciblées.
- Un environnement permettant la **gestion des dépendances** au **déploiement** et à l'**utilisation**.

Les principes

La plateforme

- Un dépôt Git local ou Gitlab communautaire pour "versionner" les fichiers (Easyconfig, Easyblock) qui décrivent comment récupérer, construire et déployer le logiciel, ainsi que ses dépendances.
- Quelques commandes et bibliothèques, principalement en python.

Les principales fonctions

- Compilation et installation automatique des logiciels souhaités, dans une version optimale pour l'architecture cible ;
- Fortement configurable : fichier, variable d'environnement ou ligne de commande ;
- Des fichiers Logs très complets ;
- Résolution automatique des dépendances ;
- Plateforme de tests unitaires et fonctionnels pour chaque "Release".

Mise en oeuvre : l'approche naïve

On installe nos logiciels.

```
[cpera@lunatic ~]$ eb -S OpenFoam
...............................
 * $CFGS1/o/OpenFOAM/OpenFOAM-5.0-20180108-foss-2018a.eb
 * $CFGS1/o/OpenFOAM/OpenFOAM-5.0-20180108-intel-2017b.eb
 * $CFGS1/o/OpenFOAM/OpenFOAM-5.0-20180108-intel-2018a.eb
 * $CFGS1/o/OpenFOAM/OpenFOAM-5.0-foss-2017b.eb
 * $CFGS1/o/OpenFOAM/OpenFOAM-5.0-intel-2017b.eb
 * $CFGS1/o/OpenFOAM/OpenFOAM-v1712-foss-2017b.eb
[cpera@lunatic ~]$ eb OpenFOAM-5.0-foss-2017b.eb --robot

[cpera@lunatic ~]$ eb -S Gromacs
...............................
 * $CFGS1/g/GROMACS/GROMACS-5.1.2-intel-2016a-hybrid.eb
 * $CFGS1/g/GROMACS/GROMACS-5.1.4-foss-2016b-hybrid.eb
[cpera@lunatic ~]$ eb GROMACS-5.1.4-foss-2016b-hybrid.eb --robot
```

Mise en oeuvre : on rationalise

Inconvénients de l'approche naïve

- Plus de 48h de compilation, 5Go de fichiers, plusieurs dépendances de logiciels installées dans des versions différentes ;
- Pas forcément la dernière release Easybuild, parfois impossible à réaliser dans intervention ;

Rationalisation

- On étudie les différentes chaînes de compilation et environnement numériques disponibles (plutôt le bazar), ainsi que nos logiciels cibles ;
- Sélection d'une "toolchain" particulière (GCC/intel, etc.)
- Résultats : moins de 48h de compilation, 3Go de fichiers, peu de multiples versions d'une même dépendance logiciel ;

```
[cpera@lunatic ~]$ eb --list-tool-chains
```

Mise en oeuvre : les ajustements

Quelques bonnes pratiques pour améliorer les temps de build :

- Opérations sur un système de fichier en RAM (/dev/shm).
- Modification de certains fichiers de configuration pour diminuer les temps de test des logiciels (FFTW) ou changer ses dépendances.
- Build en batch sur les noeuds de calcul.
- Gestion des différentes architectures avec les Modules.

```
1 [cpera@lunatic ~]$ eb -D OpenFOAM-5.0-foss-2017b.eb --job --robot
```

Mise en oeuvre : en vitesse de croisière, on s'amuse ...

- Uniquement les chaînes de compilation/build FOSS et INTEL dans le dépôt communautaire (meta paquet compilateur/configuration/bibliothèques numériques/mpi)
- Mise à jour régulière avec la dernière Release d'EasyBuild (rythme semestriel).
- Dépôt local pour gérer les fichiers de configuration easybuild/easyblock/easyconfig personnalisés.
- Upstream des logiciels et configurations peu maintenus.

Bilan

Limitations

- Lenteur du build ! (nécessité de parallélisation)
- Pas de prise en compte du cycle de vie du logiciel (update/uninstall/rollback).
- Une maitrise de l'environnement qui demande un gros investissement (python/Easybuild/toolchains/autotools/Cmake/...).

$\Rightarrow$ **Partage d'expertise et des recettes au sein de la FLMSN**

Bilan

Avantages

- Une optimisation des performances à l'exécution.
- Une gestion des dépendances efficaces.
- Un environnement cohérent avec quelques efforts.
- Une gestion très simple du multiversions et du multiplateformes.
- Une collection de plus de 2000 logiciels, utilisés dans le calcul scientifique.
- Une plateforme robuste et stable depuis 2016.
- Une reproductibilité des processus d'installation et un portage vers d'autre plateforme simplifiés.
- Une communauté importante, (ré-)active et beaucoup d'expertise dans le domaine du calcul scientifique.

Comparaison Kenneth Hoste FOSDEM 2018

And the winner is:

	CONDA	easybuild	GuixHPC	Nix	Spack
platforms	Linux, macOS, Windows	Linux, Cray	GNU/Linux	Linux, macOS, Unix	Linux, macOS, Cray
implementation	Python 2/3, YAML	Python 2	Scheme, Guile	C++, Nix (DSL)	Python 2/3
supp. software	> 3,500	> 2,000	< 6,500	> 13,000	> 2,300
releases, install & update	★★★½	★★★½	★★½	★★½	★★
documentation	★★★½	★★★	★★★½	★★★½	★★★½
configuration	★★★½	★★★	★★	★★	★★★½
usage	★★★	★★★	★★★	★★★	★★★
time to result	★★★★	★★½	★★★★	★★★★	★★★
performance	★★	★★★★	★★	★★	★★★
reproducibility	★★★	★★	★★★★	★★★★	★★

31